NASA Technical Memorandum 84501

Advanced Reliability Modeling of Fault-Tolerant Computer-Based Systems

Salvatore J. Bavuso

MAY 1982

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

ADVANCED RELIABILITY MODELING OF FAULT-TOLERANT COMPUTER-BASED SYSTEMS

Salvatore J. Bavuso
NASA Langley Research Center
Hampton, Virginia 23665/USA

SUMMARY

Digital fault-tolerant computer-based systems are on the verge of becoming commonplace in military and commercial avionics. These systems hold the promise of increased availability, reliability, and maintainability over conventional analog-based systems through the application of replicated digital computers arranged in fault-tolerant configurations. Three tightly coupled factors of paramount importance which will ultimately determine the viability of these systems are reliability, safety, and profitability. Reliability, the major driver, involves virtually every aspect of design, packaging, and field operations as regards safety, maintainability, and invariably profit for commercial applications or national security for military uses.

The antithesis of promise for the digital computer is, however, the Achilles' heel of the reliability engineer. The utilization of digital computer systems makes the task of producing a credible reliability assessment a formidable one. The root of the problem is embodied in the very essence that makes the digital computer such an outstanding device for use in a host of applications, namely its adaptability to changing requirements, computational power, and ability to test itself efficiently. It is the intent of this presentation to address the nuances of modeling the reliability of systems with large state sizes, in the "Markov" sense, which result from systems that are based on replicated redundant hardware and to discuss the modeling of numerous factors which can reduce reliability without concomitant depletion of spare hardware. The diminishing factors are captured by the popular "coverage" terminology. Advanced coverage (fault-handling) models are described with supporting rationale. Methods of acquiring and measuring parameters for these models are delineated, and some recently measured latent-fault data are presented.

INTRODUCTION

It is the intent of this paper to report on the development of two novel methodologies for the reliability assessment of fault-tolerant digital computer-based systems: Computer-Aided Reliability Estimation III and Gate Logic Software Simulation. Both technologies were developed to mitigate a serious weakness in the design and evaluation process of ultrareliable

digital systems. The weak link is based on the unavailability of a sufficiently powerful modeling technique for comparing the stochastic attributes of one system against others. Some of the more interesting attributes are reliability, system survival, safety, and mission success.

A long-term goal of the NASA Langley Research Center is the development of this tool. The technology development process is shown in figure 1. Historically, our interest in this subject commenced circa 1971. At that time, two math models were identified as having potential for filling the assessment gap. Figure 1 shows those models as CARE, Computer Aided Reliability Estimation, a computer program generated at NASA's Jet Propulsion Laboratory for application to long-lived, space-borne computer systems; and TASRA, Tabular System Reliability Analysis, a computer program due to Battelle Memorial Laboratories for application to the F-111 Pitch Flight Control System (refs. 1 and 2).

The CARE computer program is a very powerful reliability assessment capability for fault-tolerant system concepts that existed in the late 1960's. A major innovation in reliability modeling in CARE was the incorporation of the stochastic concept of coverage due to Roth and Bouricius, et al. (ref. 3). Coverage, defined as the conditional probability that a proper recovery occurs if a fault exists, was shown by Bouricius and Carter, et al., to be a significant factor for achieving high reliability in modular replacement systems (ref. 4). Prior to this consideration, reliability analyses omitted the coverage parameter entirely which caused math models to assume a unity probability of system recovery given a fault occurrence, thereby forcing the reliability predictions to be nonconservative and hence, inaccurate. Although powerful and innovative, the CARE math model suffers from two major deficiencies, its inflexibility to model the emerging multiprocessor-based systems and the lack of a model for computing the coverage parameter.

The TASRA computer program, in contrast to CARE, utilizes the popular "Markov" analysis method which allows a very flexible modeling technique but lacks the vital coverage model as well.

Based on these findings, NASA-Langley participated in the codevelopment of CARE II with the Raytheon Company (refs. 5 and 6). The primary objective in creating CARE II was to develop a coverage model to compute coverage for CARE. Figure 2 presents the coverage math model and delineates the factors comprising the coverage computation. Although the CARE II coverage model represents a quantum leap in coverage modeling, CARE II still retains the original CARE system's architectural-description inflexibility.

A gestation period ensued following the CARE II development that involved Langley in numerous studies, depicted in figure 1 as square blocks, and the codevelopment of two new reliability assessment methodologies, i.e., CAST (Combined Analytic Simulative Technique) and CARSRA (Computer-Aided Redundant System Reliability Analysis). The coverage impact study determined upper and lower bound values of coverage for a fault-tolerant triplex flight control computer system utilizing state of the art hardware (ref. 7). The CAST study made two important contributions to our program at Langley: it emphasized the potential importance of transient modeling in

reliability predictions, and it introduced the notion of combining an analytical approach with computer simulation (ref. 8). By partitioning the modeling of ultrareliable systems in this latter manner, an otherwise intractable problem using either technique in toto, now becomes workable. The CAST concept has become the mainstay of our approach to reliability assessment since the completion of the CAST study, circa 1974.

CARSRA was a spin-off from a Boeing Company study on the design of an Airborne Advanced Reconfigurable Computer System (ARCS)(ref. 9). The development and application of CARSRA was our first comprehensive involvement with the assessment of complicated aircraft flight control systems. The complexity of flight control systems gave rise to the creation of CARSRA in two important areas. Since CARSRA utilizes the Markov approach, a state reduction technique was required, and it became clear that the assessment technique must model stage dependencies in order to assess a variety of system configurations which constitute continued mission success. The most familiar example of this application is system survival. In a redundant fault-tolerant system, there are many system configurations which will effect the proper system output; however, there are other system configurations that may be of interest in addition to system survival. Boeing, in the ARCS study, defined the term "Functional Readiness." It is expressed as a time-dependent probability and is applied to missions containing critical subtasks which will either be performed or not performed, depending on the operational redundancy level at the time of demand. Boeing cites, as an example, an aircraft automatic landing function for which a certain level of hardware redundancy is required before a landing may be initiated in poor visibility and weather conditions. CARSRA also benefited from its predecessors by incorporating a multi-coverage parameter capability and an electrical transient modeling capability.

Transient modeling proceeded in two directions: the stochastic estimation of intermittent failures of computer piece-parts and the modeling of the effects of induced analog transients on digital circuitry (refs. 10 and 11). The latter study is ongoing work that relates an analog transient source with a digital system's activity. The form of this relationship will be a stochastic model for input to a system reliability assessment model.

Software reliability studies are an ongoing activity at Langley. The most recent completed study suggests the possibility of estimating software reliability through testing. Although still in the experimental stages, a methodology has been proposed and demonstrated that estimates probability of software error as a function of execution time and test trials (ref. 12).

All of the activities depicted by figure 1 have culminated to form the basis for the development of CARE III. CARE III was codeveloped by Langley and the Raytheon Company and cosponsored by the U.S. Air Force Avionics Laboratory at the Wright-Patterson Air Force Base (refs. 13, 14, and 15). A summary of the salient features of CARE III is shown in figure 3.

CARE III - A General-Purpose Reliability Analysis and Design Tool for Fault-Tolerant Systems

CARE III was designed to model large ultrareliable replicated systems incorporating digital electronics. Examples of such systems are shown in figure 4 (refs. 16, 17, 18, and 19). The CARE III assessment process is depicted in figure 5 and begins with an architectural description of an ultrareliable system. That description may be based on a conceptual model of the system, then CARE III is used as a design tool; or, the system may be well-defined so that CARE III is utilized as an analysis tool. In either case, the analyst generates a set of failure rates and probability density functions for the various failure and error mechanisms he wishes to include in the analysis. A partial list of failure and error models is delineated in figure 12. The inclusion of any of these models will necessarily lower the system reliability estimate. The need to include a model is of course a function of the architectural structure, its fault-handling mechanisms, and the magnitudes of the parameters in the model. The large choice of failure and error models is provided to increase the realism and credibility of the analysis. The models are user options in CARE III and may be omitted at the discretion of the analyst. Usually he will omit certain models after determining that they have a minimal effect. Some of this modeling information is used to define the system fault-handling model(s) which is required as a user input, indicated by the state diagram in figure 5. The remainder of the failure and error modeling data is entered as Fortran NAMELIST statements - examples will follow.

Another important step in setting up CARE III input is the generation of the system configuration and success criteria. Fault-tolerant systems are usually designed to have many hardware and functional combinations that enable proper system operation. CARE III uses the powerful fault tree language to describe system failure configurations. In large systems, the number of success combinations can be very large, and for this reason CARE III uses the "unsuccess" or failure combinations instead. In a properly designed system, the number of failure combinations should be considerably less than the number of system success combinations, thus easing the computational task. The fault tree language provides an excellent medium for delineating the system failure combinations.

Referring again to figure 5, the user-prepared data are initially processed by the CARE III input subprogram, CAREIN, shown as the upper disk. It is essentially composed of the fault tree language program. The second CARE III subprogram, COVRGE, processes the fault-handling model(s) data and puts them into the form required by the third subprogram, CARE3, which performs the reliability computations. CARE III is written entirely in the Fortran IV language and currently executes on the Digital Equipment Corporation PDP-10 computer and on the Control Data Corporation CYBER 170 series computers. The CARE III output data take two forms, graphical or tabular. In either case, the outputs of most interest are the total system reliability or system survival as a function of time and two vital components: probability of system failure due to hardware redundancy limitations (exhaustion of spares) and probability of system failure due to improper fault handling. In ultrareliable systems, the latter factor is the predominate cause of system failure (ref. 20).

An example of the CARE III assessment process is given by figures 6 and 7. Figure 6 is a sketch of an Ultrareliable Fault-Tolerant Multiprocessor composed of 10 memory processor pairs which communicate with each other over 5 individual bus lines shown in the chart as a solid bus line. This system survives if at least 2 computers and 2 buses are operational. The analyst wants to compute the probability of system survival at 10 hours of mission time for this multiprocessor system and the probability of system failure at 10 hours due to spare hardware depletion and due to improper system-fault handling. For this simple illustrative example, the analyst creates two fault trees and a state diagram for system-fault handling as depicted in figure 7. The System Fault Tree describes the system stage configurations that cause system failure. The computer stage is comprised of up to 10 computers, each having an identical failure rate. The bus stage is composed of up to 5 buses, each having an identical failure rate, most likely different than that of a computer. The OR gate in the System Fault Tree means that the system fails if a computer stage fails or a bus stage fails. A computer stage fails if less than 2 computers are operational, and a bus stage fails if less than 2 buses are operational. These conditions are described in the line beginning with "$STAGES". This statement is a Fortran NAMELIST statement. It says there are two stages, (NSTGES=2),i.e., 10 computers and 5 buses, (N=10,5), and the minimum for stage survival is 2 for each stage (M=2,2). The remainder of the line describes the form of output data requested. The fault tree description for CARE III input is shown under the heading, SYSTEM FAULT-TREE. It describes the gate interconnections and the types of gates. There is another cause of system failure that is implicit in the SYSTEM FAULT TREE and that is system failure due to single-point failures in either stage. Details of this model are discussed later.

A unique modeling capability of CARE III is the incorporation of the effects of synergistic pairs of failures. In fault-tolerant systems, the system could contain many undetected (latent) failures which individually would not cause system failure; however, certain groupings of failures that coexist may bring the system down. The CRITICAL-PAIR TREE enables the analyst to specify the conditions under which synergistic paired failures cause system failure. For this case, any two latent computer failures out of ten computers or any two latent bus failures out of five buses cause system failure. In practice, one would usually specify which paired failures cause system failure. The CRITICAL-PAIR TREE is described by the data listed under the heading, CRITICAL-FAULT PAIRS. The next step in the CARE III input process is the description of the FAULT-HANDLING MODEL. This simple state model is composed of two system states, active (A) and active detected (AD). The active system state is entered when a failure occurs. It is an undetected or latent state. If a fault detector is employed, δ is the rate at which failures are purged from the system. CARE III assumes that if the system enters the active detected state and it has spare hardware, it will reconfigure out the faulty module and the system recovers. Note, as δ increases, the probability of synergistic failures occurring diminishes since there will be fewer latent failures present. Line one, $FLTTYP, shows that there is one fault model (NFTYPS=1) and defines the value of δ as 3.6×10^2 detections per hour. Line 3, $FLTCAT, simply links failure rates denoted as RLM arrays to stages (JTYP). Line 5, $RNTIME, specifies flight time of 10 hours. CARE III input data for this

example system is shown in figure 7 beginning with the statement $FLTTYP and including all the statements that follow. The CARE III output is the total system probability of failure, the system probability of failure due to improper fault handling, and the probability of system failure due to spares exhaustion.

User-Oriented Language for Describing Complex System Failure Configurations (Fault Tree)

The multiprocessor example made use of a trivial application of the CARE III fault tree language. A better example emphasizing the power of the fault tree input is given by figures 8 and 9. Figure 8 shows a block diagram of a proposed fault-tolerant flight-control system. Of particular interest is the Pitch Augmentation Stability (PAS) short cycle function. The system fault tree for this function is presented in figure 9. This tree illustrates that not only hardware redundancy can be represented but functional redundancy as well. The elevator math model is functionally redundant to the secondary actuators. The melding of hardware and functional redundancy is a common practice in aircraft design. The proper entry of this fault tree into CARE III with the necessary failure rate and fault-handling data would yield a prediction of the probability of loss of PAS function as a function of mission time. For the uninitiated, figure 9 is read as follows: An output from logic OR gate 212 constitutes loss of PAS function which can occur if an output from OR gate 211 occurs, or if an output from gate 210 occurs, or both. Gate 210 yields an output if at least 3 out of 4 secondary actuators or actuator function (elevator math model) fail. Secondary actuator A will fail if computer A fails, or actuator A fails, or both. A similar description can be used to delineate failures due to loss of computation or loss of sensors.

Fault-Handling Model Based on Probabilistic Description of Operative Detection, Isolation, and Recovery Mechanisms

In figure 7, a simple fault-handling model of two states was described. CARE III has both a single-fault model and a double-fault model. The latter defines critically coupled, paired failures. The single-fault model is given in figure 10 and is shown in the dashed box. For illustrative purposes, three additional states have been added so that the state diagram is a Markov model of a 2 unit system. Initially, the system is in state 0 and has experienced no failures. When a failure occurs, the system enters state A, the Active latent state, given by the arrival density, $\lambda(t)$. Depending upon the nature of the failure, i.e., permanent, transient, intermittent, etc., the fault-handling model will be defined differently. For example, if the failure were intermittent, $\lambda(t)$ would be the probability density function (PDF) for the arrival of an intermittent, and states A and B define the intermittent model where α and β are constant transition rates into and out of state B, respectively. When the system is in state B, the Benign state, the failed unit appears to have healed itself, i.e., the manifestation of the failure, a fault, vanishes;

however, when the failed manifestation is once again resumed (the fault reappears), the system enters state A where at that point, the failure looks like a hard failure. It could be detected by a self-test program with PDF $\delta(t')$, and the system would enter state AD, the Active Detected state; where given that a spare exists, the system will purge the faulty unit and switch in the spare. Or while in the active state, the fault could generate errors with PDF $\rho(t')$. The system then will enter the AE, Active Error state. The intermittent failure could manifest its intermittent state again so the system would then enter state BE, the Benign Error state. Although the failure is benign, the error may not be benign and may cause system failure which is denoted by the BE to F transition $[(1-C)\varepsilon(\tau)]$. The error detection density $\varepsilon(\tau)$ and 1-C is the proportion of errors from which the system is unable to recover. While in state BE, the error could be detected and corrected. In this event, the system enters state BD (Benign Detected) by transition $C\varepsilon(\tau)$. At this point, the system may choose to do nothing further with the detected error and so move to the Benign state, or the system may choose to reconfigure out the module containing the error and, therefore, move to state 1. The other transition out of state AE is to state F, the single point failure transition $[(1-C)\varepsilon(\tau)]$. This transition is similar to the BE to F transition. In a well-designed fault-tolerant system, $(1-C)\varepsilon(\tau)$ should be near zero in magnitude. If $\lambda(t)$ were the PDF for the arrival of a transient, α would be set to a value greater than zero and β would be equal to zero. The PDF $\lambda(t)$ for the arrival of a permanent failure would be defined so that $\alpha=\beta=0$. The dashed arc going from state AD to A, enables the analyst to include the effects of the system decision that the detected fault which took the system from state A to AD was, in fact, a transient. In this regard, the system would not reconfigure out a non-failed module. A judicious choice of values for the single-fault model affords the analyst a wide range of models. A different fault model may be assigned to each stage or several models may be assigned to a given stage to cover the effects of different failure mechanisms such as transients, intermittents, hard failures, etc.

The reader will note that the reliability model in figure 10 has three measures of time associated with it which necessarily makes the model a semi-Markov process. This added complexity is required because the behavior of the system is dependent on the onset of the various fault-behavior events.

LARGE REDUCTION OF SYSTEM STATE SIZE

The capability thus described comes at no small computational price if state of the art techniques were employed. In fact, if one were to utilize the popular "Markov" modeling technique on a nontrivial system such as the flight control system shown in figure 11 (which is composed of 22 stages and 64 reconfigurable modules) coupled with a reasonable set of failure and error models (some of which are delineated in figure 12), the number of system states would be on the order of millions. For each state, a linear differential equation is formed. Clearly the solution of millions of differential equations is computationally intractable if not impossible with today's technology. But CARE III was designed to assess these types of systems. How does it do it?

To understand the CARE III state reduction method, it is expedient to first examine how the state size build-up occurs in the Markov method. A Markov state is described as an ordered n-tuple. The components of the n-tuple contain information about the number of failed reconfigurable units in the system plus system fault-handling information for each module and fault type (hard failure, transient, etc.). For the system shown in figure 11, the n-tuple has a minimum of 22 components, i.e., one for each stage. For each stage, additional n-tuple fault-handling components are added to describe the number of failed units that are system detected, the number that are identified with a reconfigurable module, and the number that have been recovered. A set of fault-handling components is included in the n-tuple for each type of failure, e.g., transient, hard, intermittent, etc. The total number of n-tuple components becomes very large. The product of the n-tuple components gives the number of possible system states. In contemporary practice, tractable analyses are accomplished by making numerous assumptions about the system to reduce the state size to the order of 1000. CARE III, on the other hand, retains a considerable amount of detail without the burden of unmanageable state sizes. This feat is accomplished in CARE III by separating fault-handling information from the structure model, i.e., information about the number of failed units. Each model is worked separately to a point and then recombined (ref. 21). An example of this state reduction is depicted by figures 10 and 13. When CARE III processes the fault-handling model of figure 10, that information is mapped into time-varying transition rates, $\lambda_1(t)$, $\lambda_2(t)$, as shown in figure 13. What might have been a stationary semi-Markov process for the system of figure 10 will always become a nonstationary Markov process. For large systems, state size reductions of at least 10,000 to 1 have been estimated. The solution to the nonstationary process model of figure 13 is given by the solution to the forward Kolmogorov equation depicted in figure 14. The system reliability is computed by summing the probabilities, $P_\ell(t)$, for the allowable or success states. Numerically, it is more accurate to compute the probability of system failure in lieu of reliability (probability of system survival). The user-defined fault trees specify the system failure states, so that, the probability of system failure is simply the sum of $P_\ell(t)$ over ℓ, the set of system failure states. CARE III actually computes the probability of system failure using the equation,

$$Q_\ell(t) = e^{-\int_0^t \lambda_\ell(\tau)\,d\tau} \int_0^t \frac{\sum_{j \neq \ell} \left[Q_j(\tau) + P_j(\tau)\,\bar{c}_{j\ell}(\tau) \right] \lambda_{j\ell}(\tau)}{e^{-\int_0^\tau \lambda_\ell(\eta)\,d\eta}}\,d\tau$$

where the probability of system failure is given by the sum of $Q_\ell(t)$ over ℓ, the set of system failure states.

CARE III has entered the first stage of validation by undergoing extensive testing at the computer program (debugged) and math model levels. It is also being applied to several experimental ultrareliable design concepts to evaluate CARE III modeling flexibility and the user-oriented fault tree interface.

GLOSS-GATE LOGIC SOFTWARE SIMULATION

It is one thing to implement a very powerful reliability model and quite another to make it useful. For all reliability evaluators, including CARE III, a weakness lies in the unavailability of data for many of the fault-handling parameters. The situation is not a total loss; however, since reasonable engineering estimates can be made in many cases, and furthermore, the sensitivity of the system reliability can be tested against variations in the marginal data. A better way, of course, is to measure or estimate the parameters based on some empirical observations.

LATENT-FAULT MODELING AND MEASUREMENT METHODOLOGY

Since system fault detection appears to be the most critical fault-handling parameter, NASA-Langley in 1977 initiated a series of studies to investigate a methodology for measuring the fault latency of digital computers (ref. 22). The methodology consisted of simulating a 1000 equivalent gate processor in a host CDC Cyber 173 computer. The simulated processor was a paper design and is referred to as a "hypothetical" machine. The hypothetical machine was simulated at the gate level. Actually, two copies of the hypothetical machine executed identical code in synchronism, where one machine received a stuck-at fault at the onset of the computation. Detection or nondetection was determined after the nonfaulted processor completed its execution. At that time, the computational results of the two simulated machines were compared, bit for bit. Any difference constituted detection. If no detection occurred, the code's input variables were randomly altered, and the processes were repeated for the same fault. This scheme was repeated for up to eight executions for the same fault, if detection didn't occur. If a detection occurred in less than or equal to eight repetitions, or no detection occurred after eight repetitions, then a new trial began where another stuck-at fault was induced. This overall process was repeated for up to 1000 randomly selected faults. The 1000 induced faults were selected as a function of piece-part failure rates and were distributed equally across the nodes of the gates. The latency time, i.e., time to fault detection is expressed in number of code executions or repetitions. The time scale can easily be mapped into CPU seconds of code execution, if desired.

The comparison of output data from two or more computers is often referred to as a comparison-monitoring detector which is an important detection mechanism employed in many operational fault-tolerant systems. In the CARE III fault-handling model shown in figure 10, comparison-monitoring detection is modeled by $\varepsilon(\tau)$.

The results of the pilot study were both surprising and intriguing. Using six different programs ranging from a very simple fetch-and-store program to a very complex linear convergence scheme, the pilot study showed that only 50 percent of the induced faults were detected after eight repetitions for all six programs. Figure 15 depicts typical results. The

implication that these results have on reliability assessment for highly reliable systems is staggering. It suggests that highly reliable fault-tolerant systems cannot be designed with comparison monitoring or majority voting as the major stuck-at fault detector (ref. 7).

VERIFICATION OF LATENT-FAULT MEASUREMENT METHODOLOGY

It was with this concern that a series of further experiments to investigate the validity of the pilot study results were designed at NASA-Langley After all, it was not clear that similar results could be obtained for a real processor executing practical software. The goals of the follow-on work were to test the findings of the pilot study utilizing a real avionic miniprocessor, to assess the significance of injecting faults at the gate level and at the functional pin level, to evaluate an airborne self-test program, and to account for undetected faults (refs. 22, 23, 24, and 25). The methodology for gate level simulation, which was codeveloped by NASA-Langley and Bendix, is called the GLOSS, Gate Logic Software Simulator.

The pilot study results were tested in three phases using a gate simulation of the Bendix BDX-930 miniprocessor, a 5000 gate equivalent CPU. Initially the same six pilot study programs were coded using the comparable primitive instruction set of the hypothetical machine, i.e., load, store, add, subtract, and branch. The next phase allowed the six programs to be recoded using the rich instruction set of the BDX-930, and finally comparison-monitoring detection was measured for flight control system code in lieu of the six pilot study programs. The surprising outcome of this experiment is typified in figure 16 for all six programs. The percent of nondetected faults is about the same for all the programs, instruction sets, and two different machines, i.e., 50 percent. As the code becomes more complex, the shape of the histogram bunches up so that virtually all the detection occurs in the first execution. The latency time decreases somewhat with increased code complexity but not the percent detected.

When the same set of experiments are repeated with the exception that faults are induced at the register transfer or pin level in lieu of the gate level, similar results shown by figure 17 appear. One notable difference, however, is that the level of detection significantly rises.

As an extension to the pilot study, the latent-fault measurement methodology was applied to an airborne self-test program consisting of 2000 BDX-930 instructions which executed in three milliseconds on the BDX-930. While the simulator executed the self-test program, faults were induced at the gate level and, in a separate experiment, at the pin level. The design goal for the self-test program was 95 percent detection. Figure 18 presents a summary of the self-test detection values and the comparison-monitoring detection values. For the same level of fault inducement, the self-test code shows the highest detection, but fell short of the 95 percent requirement for gate-level faults. With considerable effort and expense, the 87 percent self-test detection was increased to 94 percent, and appears to be a practical upper bound on gate-level fault detection. Flight control system code improved fault detection substantially but still fell short of 0 percent undetected for gate-level faults. For component-level injected faults, the industry-assumed value of 0 percent undetected was achieved.

SOME PROFOUND RESULTS AND OBSERVATIONS

The wide dispersion of detection raises some confounding questions about the method of fault injection and, hence, which detection parameters to use in reliability assessments. The inducement of faults at the gate or pin levels yields a wide dispersion of detection when all other factors are equal. This concern is further exacerbated by the knowledge that the method practiced by industry, pin-level fault injection, yields the higher detection values. At our present level of understanding of fault propagation mechanisms, the pin-level detection values would appear to be nonconservative and should be used with great caution, if at all. This recommendation is based on our knowledge that the gate-level faults that were not detected after eight repetitions are potentially detectable or distinguishable, i.e., there exists some code or sequence of code execution that will propagate a distinguishable fault.

In the process of investigating the reason why faults were not detected after eight repetitions, it was discovered that there exists a class of faults that can never have an effect on the system and, therefore, can never be detected. This class of indistinguishable faults has been estimated to comprise 16 percent of all faults. An example of an indistinguishable fault is a stuck-at fault located at the unused output of a flip-flop circuit. An important outcome of this discovery regards the method of estimating detection coverage. The conservative approach, and the correct one, is to delete the 20 percent indistinguishables from the set of induced faults in the computation of detection coverage. The net effect is to reduce the magnitude of detection coverage.

The lessons learned from these latent-fault modeling and measurement studies are summarized as follows:

- Practical measurement of detection coverage for stuck-at faults is possible and is a necessary aspect of reliability assessment.
- Comparison-monitoring detection for typical application code is much less than expected, which poses serious implications for highly reliable systems.
- 95 percent gate-level self-test detection coverage is measureable and achievable but expensive to accomplish.
- The industry practice of measuring self-test detection by inducing faults at the pin level may not be conservative, and in view of the fact that the reliability of highly reliable systems is very sensitive to detection, further analysis of this practice is required.

CONCLUDING REMARKS

The CARE III and the GLOSS are presently in the developmental stages, with CARE III clearly in the lead. The CARE III math model is embodied in a Fortran IV computer program that has been receiving considerable national scrutiny. The validation of CARE III is being conducted by industry, the

university community, and by the U.S. Government at NASA's Langley Research Center and by the U.S. Air Force at Wright-Patterson Air Force Base. To date, only minor correctable problems have cropped up; and, if this trend continues, CARE III will be released within a year.

The development of a generally applicable GLOSS computer program, which embodies the GLOSS methodology, is currently underway. Initially, the GLOSS will execute on the VAX-11, 700 series computers but will be written for maximum computer portability.

REFERENCES

1. Mathur, F. P.: Reliability Study of Fault-Tolerant Computers in Supporting Research and Advanced Development. Jet Propulsion Laboratory, Aug. 1969, Space Programs Summary 37-58, Vol.III, pp. 106-113.

2. Blazek, R. H., et al.: Demonstration of Combined Reliability Prediction and Verification Techniques to a Typical Flight Control System, Vol.I, Development and Application of Tabular System Reliability Analysis to the F-111 Pitch Flight Control System. Battelle Columbus Laboratories, AFFDL-TR-71-128, Vol.1, (Available from AF Flight Dynamics Lab., Wright-Patterson AFB), Oct. 1971.

3. Roth, J. P., et al.: Phase II of an Architectural Study for a Self-Repairing Computer. SAMSO TR-67-106, U.S. Air Force, Nov. 1967. (Available from DDC as AD 825460).

4. Bouricius, W. G., et al.: Reliability Modeling Techniques and Trade-Off Studies for Self-Repairing Computers. RC 2378, Res-Div., IBM Corp., Feb. 1969.

5. Raytheon Company, Sudbury, MA: Reliability Model Derivation of a Fault-Tolerant, Dual, Spare-Switching Digital Computer System. NASA CR-132441, 1974.

6. Raytheon Company, Sudbury, MA: An Engineering Treatise on the CARE II Dual Mode and Coverage Models. NASA CR-144993, 1976.

7. Bavuso, S. J.: Impact of Coverage on the Reliability of a Fault Tolerant Computer. NASA TN D-7938, 1975.

8. Ultra-Systems, Inc., Newport Beach, CA: Reconfigurable Computer Systems Study. NASA CR-132537, 1974.

9. Bjurman, B. E., et al.: Airborne Advanced Reconfigurable Computer System (ARCS). The Boeing Commercial Airplane Company. NASA CR-145024, 1976.

10. O'Neill, E. J.; and Halverson, J. R.: Study of Intermittent Field Hardware Failure Data in Digital Electronics. Sperry Univac Defense Systems, St. Paul, MN. NASA CR-159268, 1980.

11. Masson, G. M.: Executive Summary - Intermittent/Transient Faults in Computer Systems. The Johns Hopkins University. NASA CR-159229, 1979.

12. Nagel, P. M.: Software Reliability: Repetitive Run Experimentation and Modeling. Boeing Computer Services Company. NASA CR-165836, 1982.

13. Stiffler, J. J., et al.: CARE III Final Report Phase 1, Vols. 1 and 2. Raytheon Co., Sudbury, MA. NASA CR-159122 and NASA CR-159123, 1979.

14. Bavuso, S. J.: Trends in Reliability Modeling Technology for Fault Tolerant Systems. AGARD Conf. Proc. No. 261 on Avionics Reliability, Its Techniques and Related Disciplines, April 1979.

15. Stiffler, J. J.; and Bryant, L. A.: CARE III Phase II Report, Mathematical Description. Raytheon Co., Sudbury, MA. NASA CR-3566, 1982.

16. Hopkins, A. L.; and Smith, T. B.: The Architectural Elements of a Symmetric Fault-Tolerant Multiproccesor. IEEE Trans. on Computers, Vol. C-24, No.5., 1975.

17. Osder, S.: The DC-9-80 Digital Flight Guidance System's Monitoring Techniques. Sperry Flight Systems, Phoenix, AZ, AIAA Paper 79-1704, 1979.

18. O'Hern, E. A.: Space Shuttle Avionics Redundancy Management. Rockwell International, AIAA Digital Avionics Systems Conference, April 1975.

19. Wensley, J. H., et al.: Design Study of Software-Implemented Fault Tolerance (SIFT) Computer, SRI International, Menlo Park, CA. NASA CR-3011, 1978.

20. Stiffler, J. J.: Fault Coverage and the Point of Diminishing Returns. Journal of Design Automation and Fault Tolerant Computing, Vol.2, No.4, Oct. 1978.

21. Trivedi, K. S.; and Geist, R. M.: A Tutorial on the CARE III Approach to Reliability Modeling. Duke University, Durham, NC. NASA CR-3488, 1981.

22. Nagel, P. M.: Modeling of a Latent Fault Detector in a Digital System. Vought Corp., Hampton, VA. NASA CR-145371, 1978.

23. McGough, J. G.; and Swern, F. L.: Measurement of Fault Latency in a Digital Avionic Miniprocessor. Bendix Corp., Teterboro, NJ. NASA CR-3462, 1981.

24. McGough, J. G.; et al.: Methodology for Measurement of Fault Latency in a Digital Avionic Miniprocessor, AGARD Conf. Proc. No. 303 on Tactical Airborne Distributed Computing and Networks, June 1981.

25. Bavuso, S. J., et al.: Latent Fault Modeling and Measurement Methodology for Application to Digital Flight Controls. Advanced Flight Control Symposium, USAF Academy, Colorado Springs, CO, Aug. 1981.

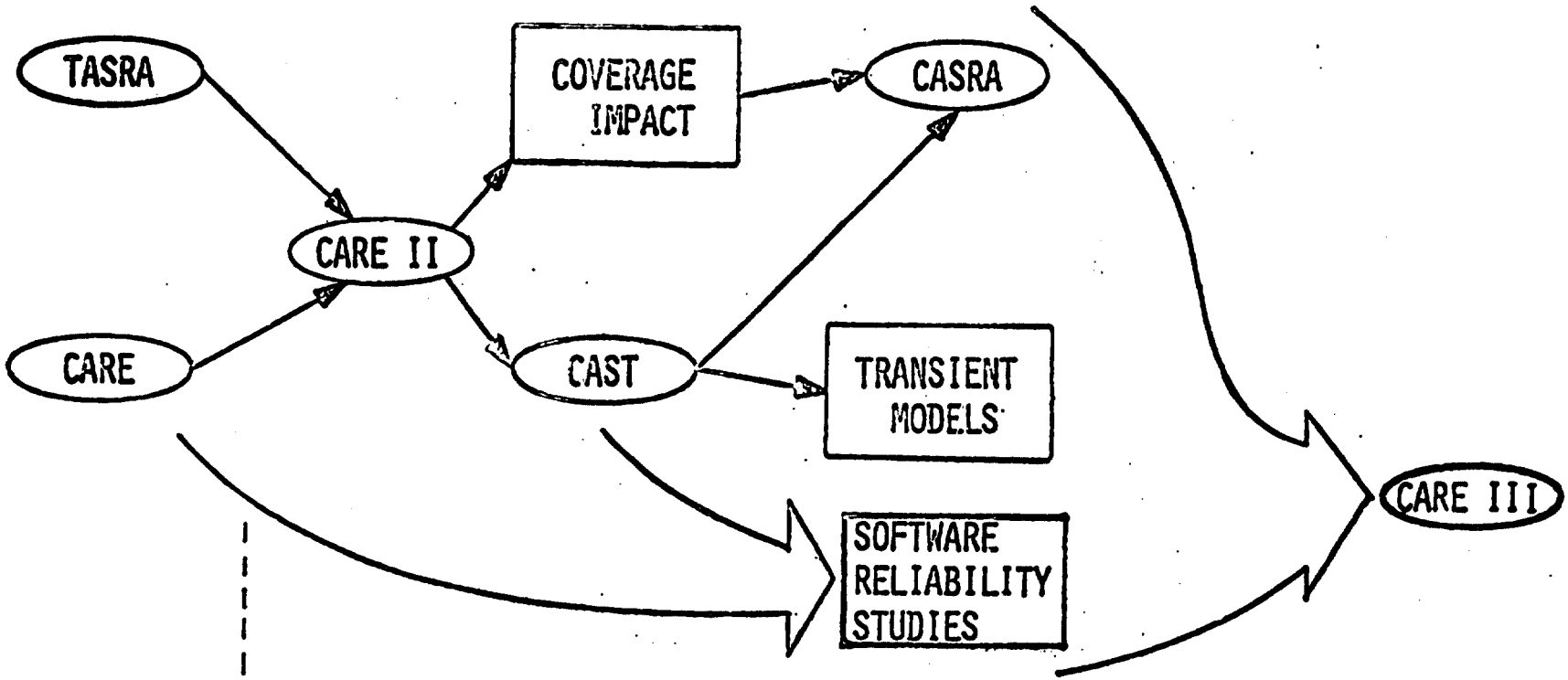


Figure 1. - Technological development leading to CARE III.

$$C_x(i,j) = P_i P_{sx}^{j-1} P_i' \int_0^\infty \int_0^\infty g_i(\tau) h_i(\tau' - j\tau_{sx}) r_i(\tau,\tau') d\tau d\tau'$$

WHERE:

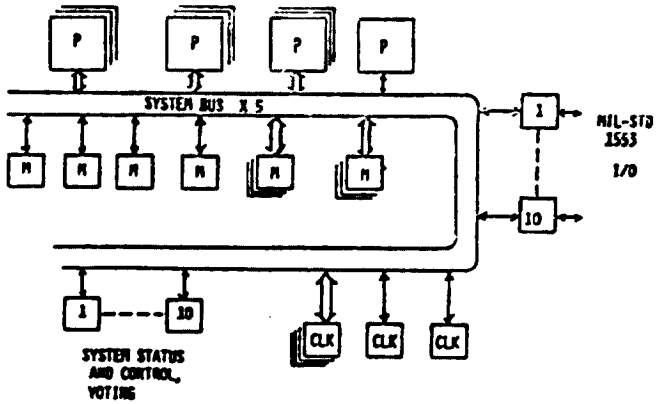
- $C_x(i,j)$ = COVERAGE TERM
- τ = DETECTION TIME
- τ' = ISOLATION TIME
- P_{sx} = DEFECTIVE SPARE DETECTION PROBABILITY
- τ_{sx} = SPARE UNIT TEST TIME
- P_i = NON-COMPETITIVE DETECTION PROBABILITY
- P_i' = ISOLATION PROBABILITY ASSOCIATED WITH P_i
- h_i = ISOLATION RATE
- r_i = RECOVERY PROBABILITY
- g_i = COMPETITIVE DETECTION RATE

Figure 2. - CARE II coverage model.

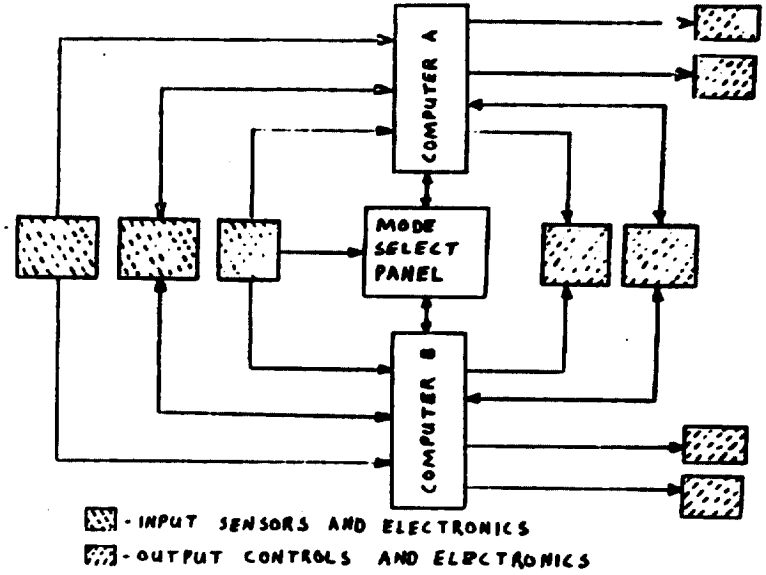
CARE III

- GENERAL-PURPOSE RELIABILITY ANALYSIS AND DESIGN TOOL FOR FAULT-TOLERANT SYSTEMS
- LARGE REDUCTION OF STATE SIZE
- FAULT-HANDLING MODEL BASED ON PROBABILISTIC DESCRIPTION OF DETECTION, ISOLATION, AND RECOVERY MECHANISMS
- VARIETY OF FAULT AND ERROR MODELS (STATIONARY AND NONSTATIONARY)
  - PERMANENT
  - TRANSIENT
  - INTERMITTENT
  - DESIGN FAULTS
  - SOFTWARE ERRORS
  - LATENT FAULTS
- USER-ORIENTED LANGUAGE FOR DESCRIBING COMPLEX SYSTEM CONFIGURATIONS AND SUCCESS CRITERIA (FAULT TREE)

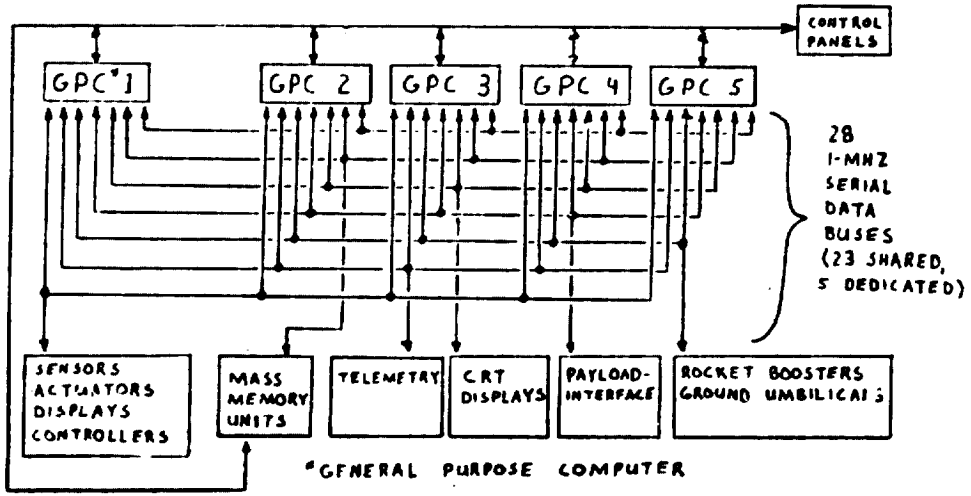
Figure 3. - CARE III salient features.



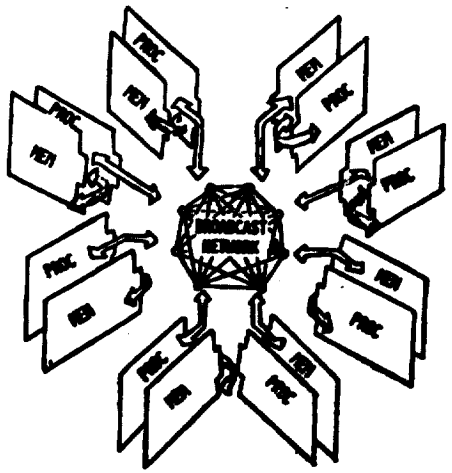
FAULT-TOLERANT MULTIPROCESSOR



DC-9-80 DIGITAL FLIGHT GUIDANCE SYSTEM



SPACE SHUTTLE SYSTEM



SOFTWARE IMPLEMENTED FAULT TOLERANCE (SIFT)

Figure 4. - CARE III applicable highly reliable fault-tolerant systems.

COMPUTER AIDED RELIABILITY ESTIMATION (CARE III)

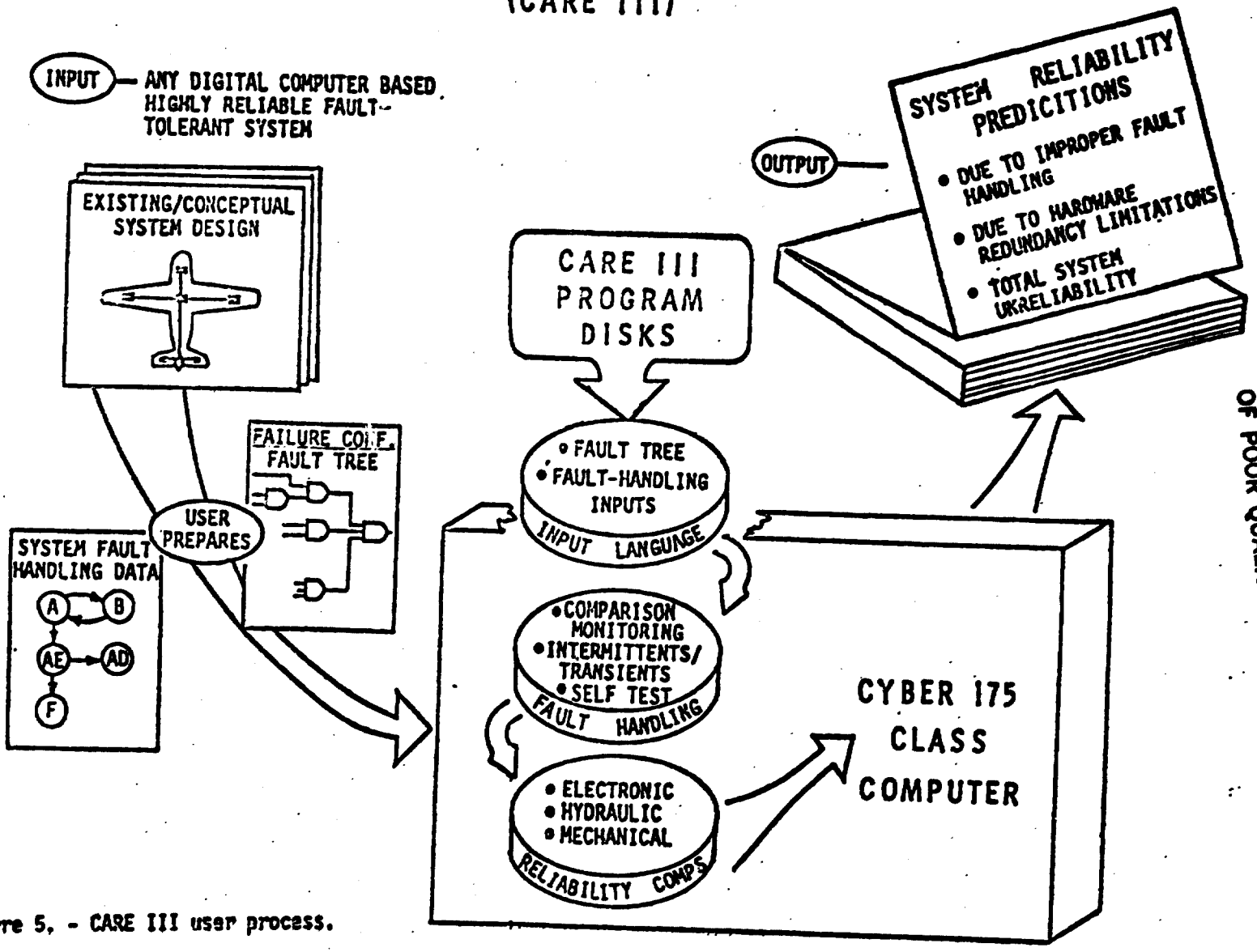


Figure 5. - CARE III user process.

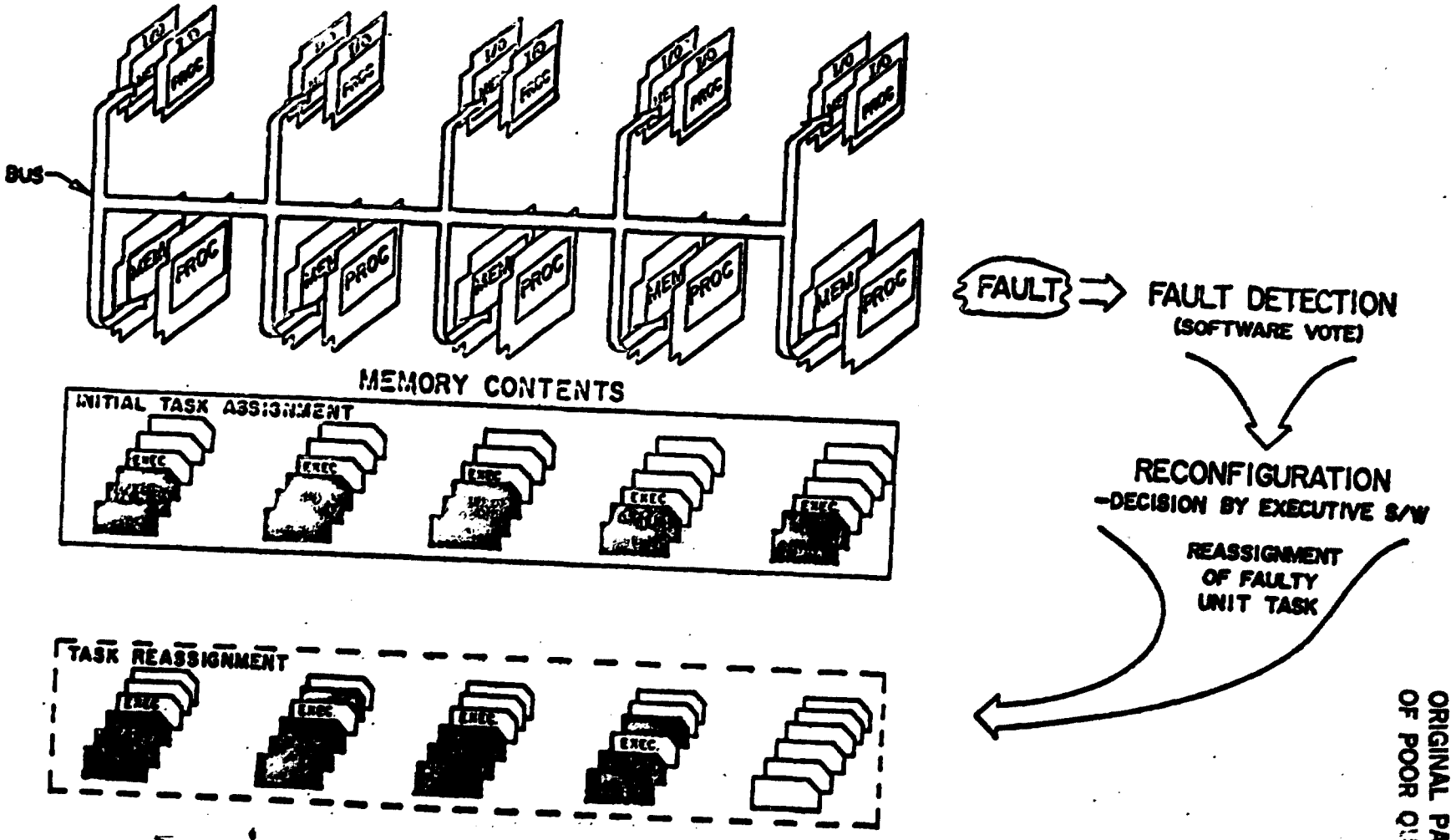
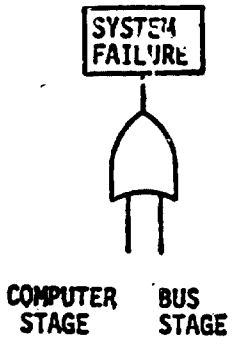
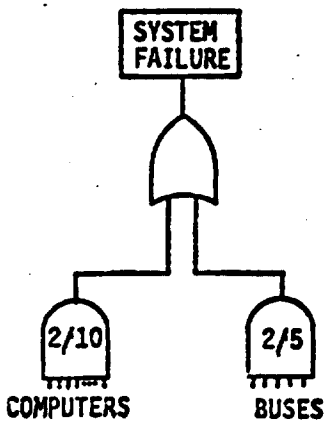


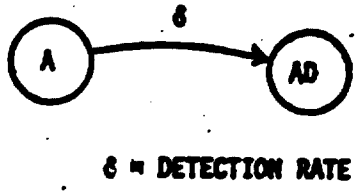
Figure 6. - Ultrareliable fault-tolerant multiprocessor.



SYSTEM FAULT TREE



CRITICAL-PAIR TREE



FAULT-HANDLING MODEL

```
$FLTTYP  NFTYPS=1,DEL=3.6E2 $
$STAGES  NSTGES=2,N=10,5,M=2,2,IRLPCD=4,RLPLOT=T$
$FLTCAT  NFCATS=2*1,JTYP(1,1)=1,JTYP(1,2)=1,
         RLM(1,1)=1.0E-4,RLM(1,2)=1.0E-5$
$RNTIME  FT=10. $
SYSTEM FAULT-TREE
1 2 3 3
3 0 1 2
      CRITICAL-FAULT PAIRS
1 15 16 18
1 1 10
2 11 15
16 2 1 2 3 4 5 6 7 8 9 10
17 2 11 12 13 14 15
18 0 16 17
```

Figure 7. - CARE III input for ultrareliable fault-tolerant multiprocessor.

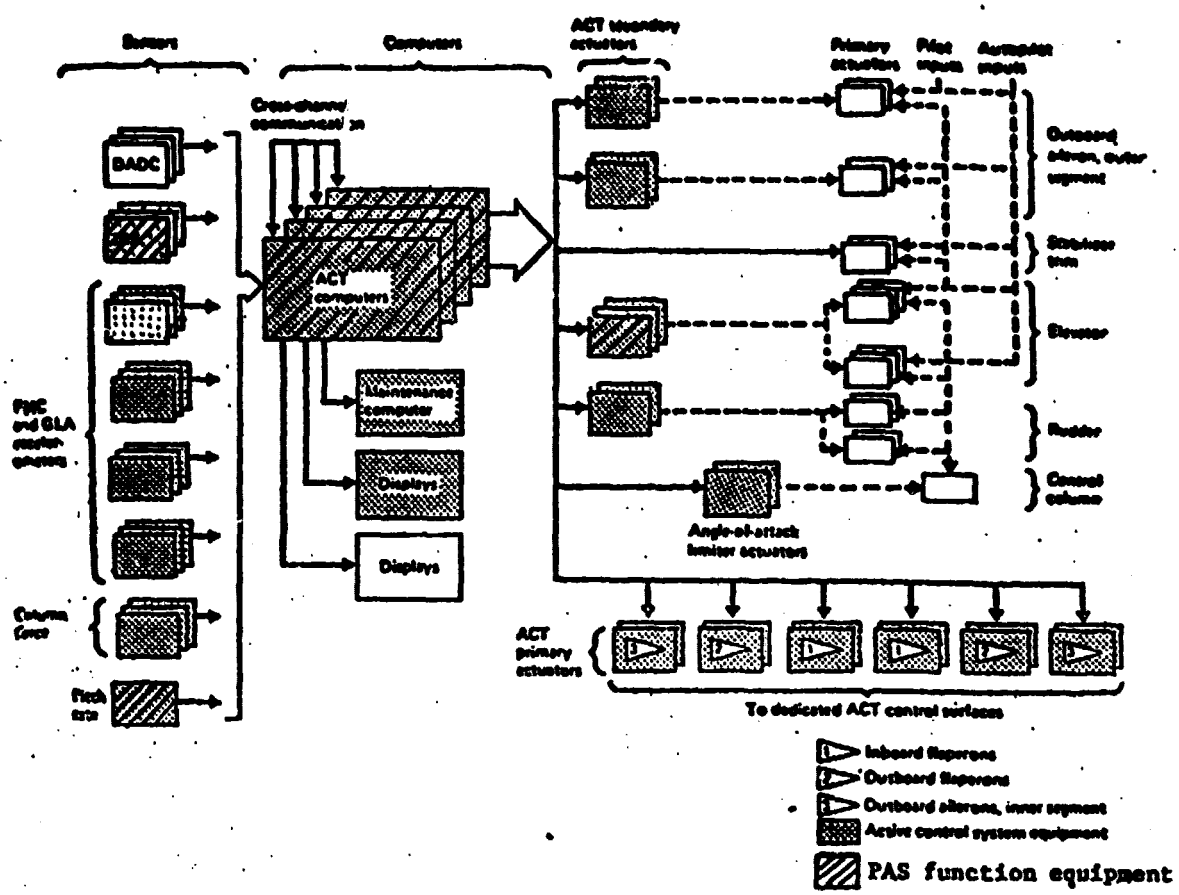


Figure 8. - Fault-tolerant flight control system - pitch augmented stability function (PAS).

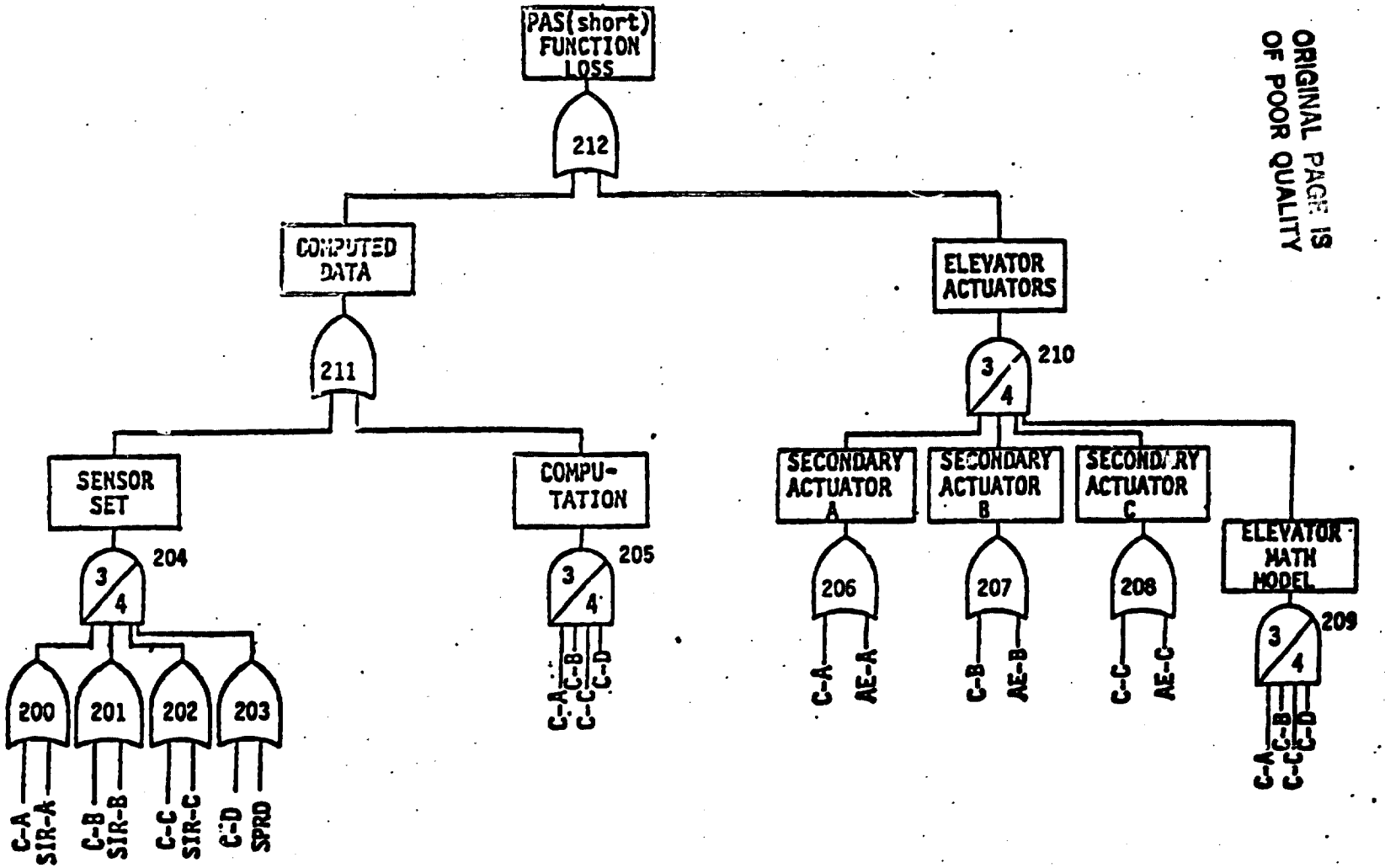
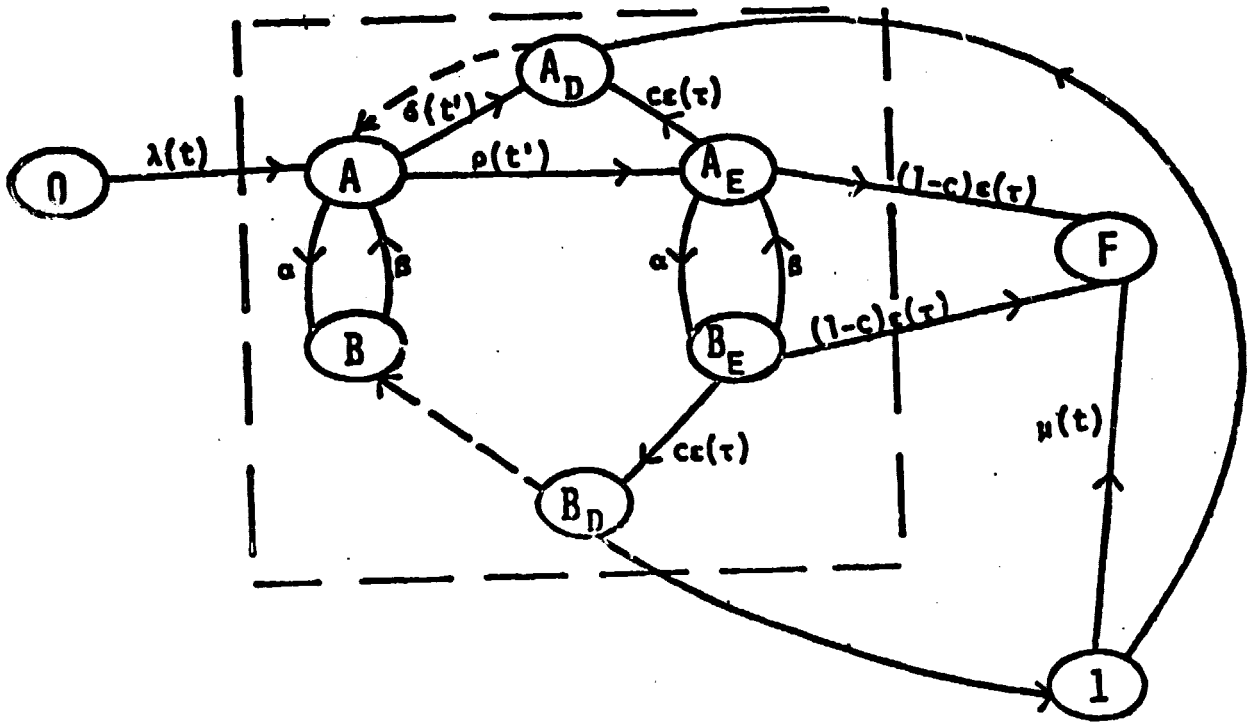


Figure 9. - CARE III fault tree input for PAS function.



t = global or mission time

t^1 = time from entry to state A

τ = time from entry to state A_E

Figure 10. - Overall reliability model of two-unit system.

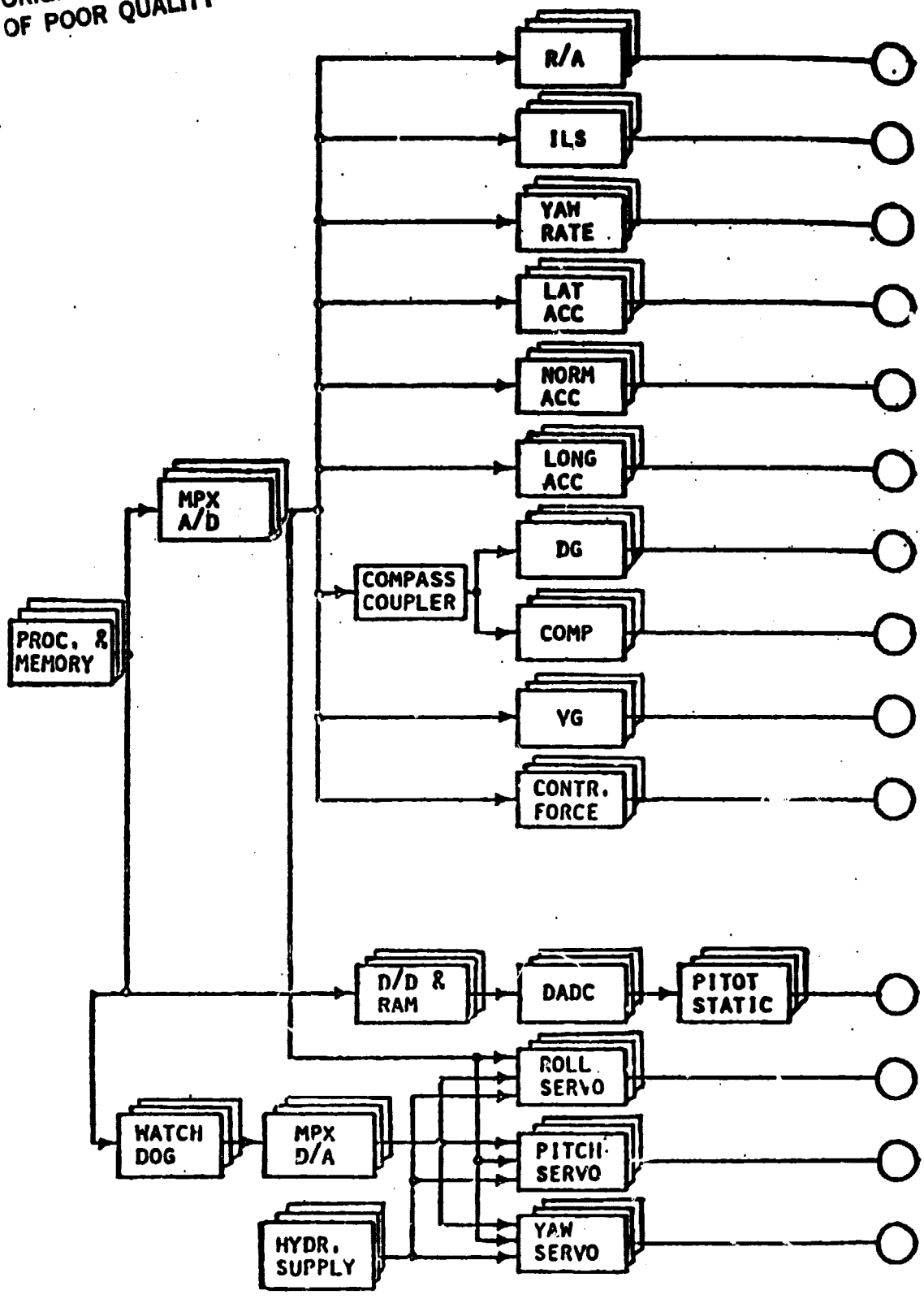


Figure 11. - Advanced reconfigurable flight control system.

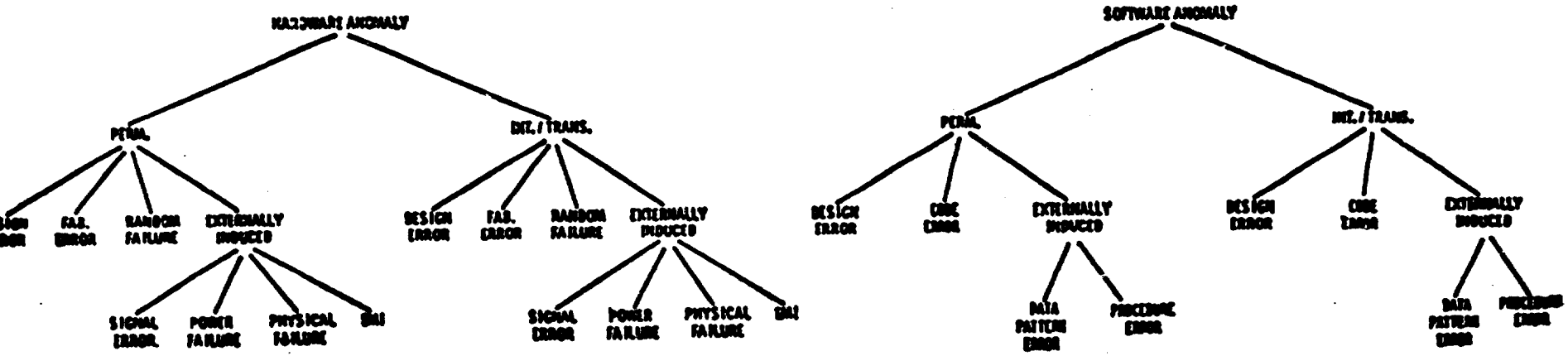


Figure 12. - Delineation of hardware and software failure and error models.

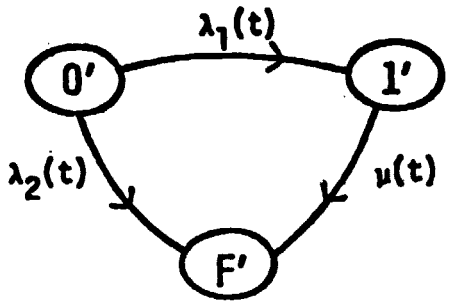


Figure 13. - Aggregated reliability model for a two-unit system.

$$P_\ell(t) = e^{-\int_0^t \lambda_\ell(\tau)d\tau} \int_0^t \frac{\sum_j P_j(\tau) c_{j\ell}(\tau) \lambda_{j\ell}(\tau)}{e^{-\int_0^\tau \lambda_\ell(\eta)d\eta}} d\tau$$

where

$P_\ell(t)$ = probability of being in state ℓ at time t

$\lambda_{j\ell}(t)$ = transfer rate from state j to state ℓ

$\lambda_\ell(t) = \sum_j \lambda_{\ell j}(t)$

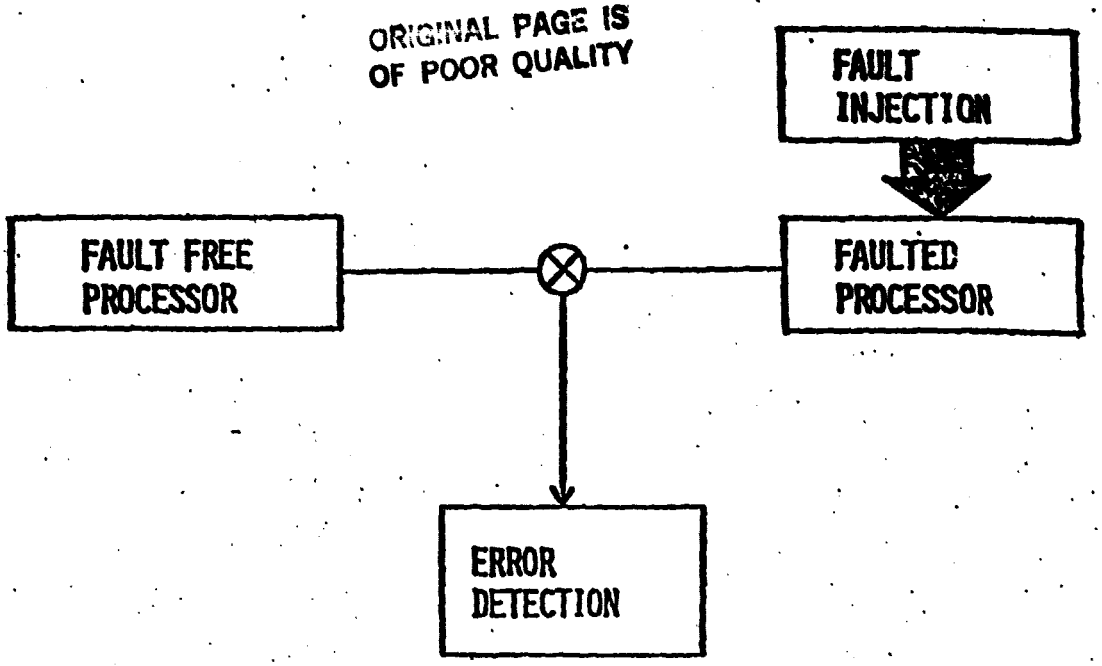
$c_{j\ell}(t)$ = coverage associated with a failure which, if coverage were perfect, would cause a transfer from state j to state ℓ

The system reliability is given by

$$R(t) = \sum_{\ell \in L} P_\ell(t)$$

for the set L of allowable states.

Figure 14. - CARE III state probability computation.



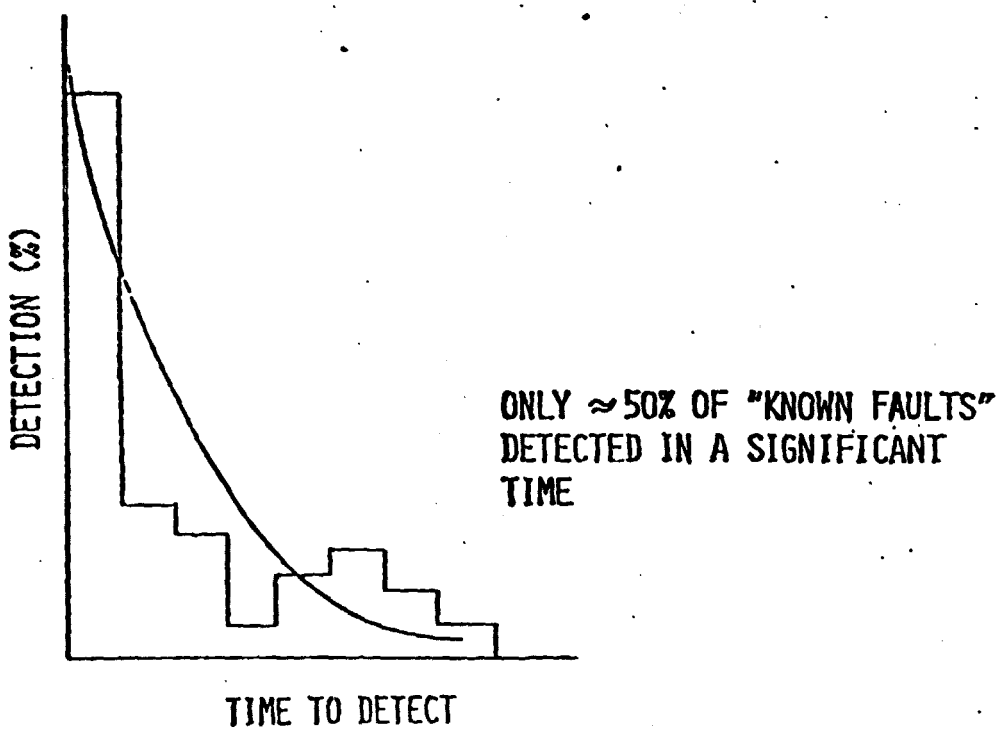


Figure 15. - Latent-fault measurement.

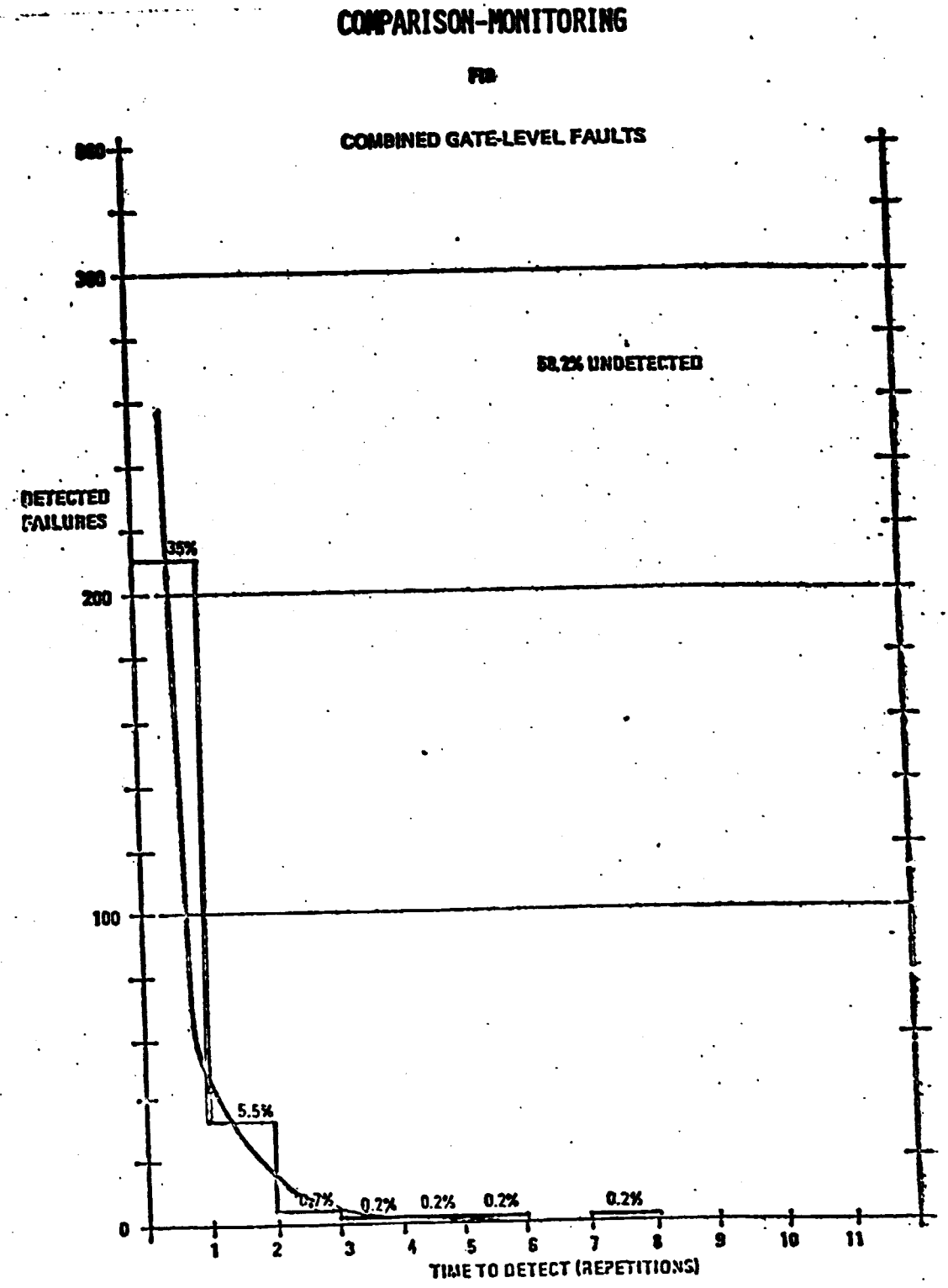


Figure 16. - Fault latency distribution - gate-level faults.

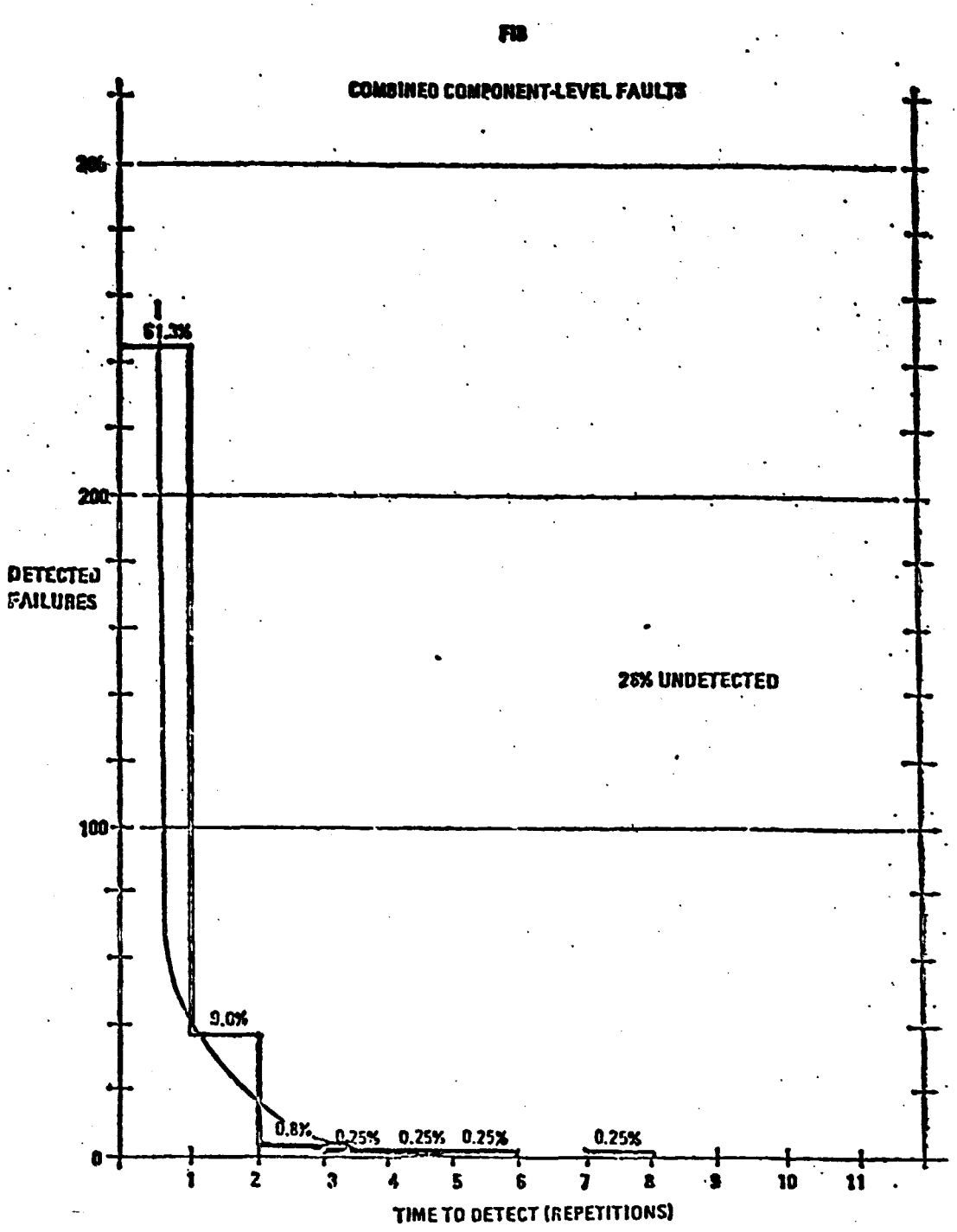


Figure 17. - Fault latency distribution - component-level faults.

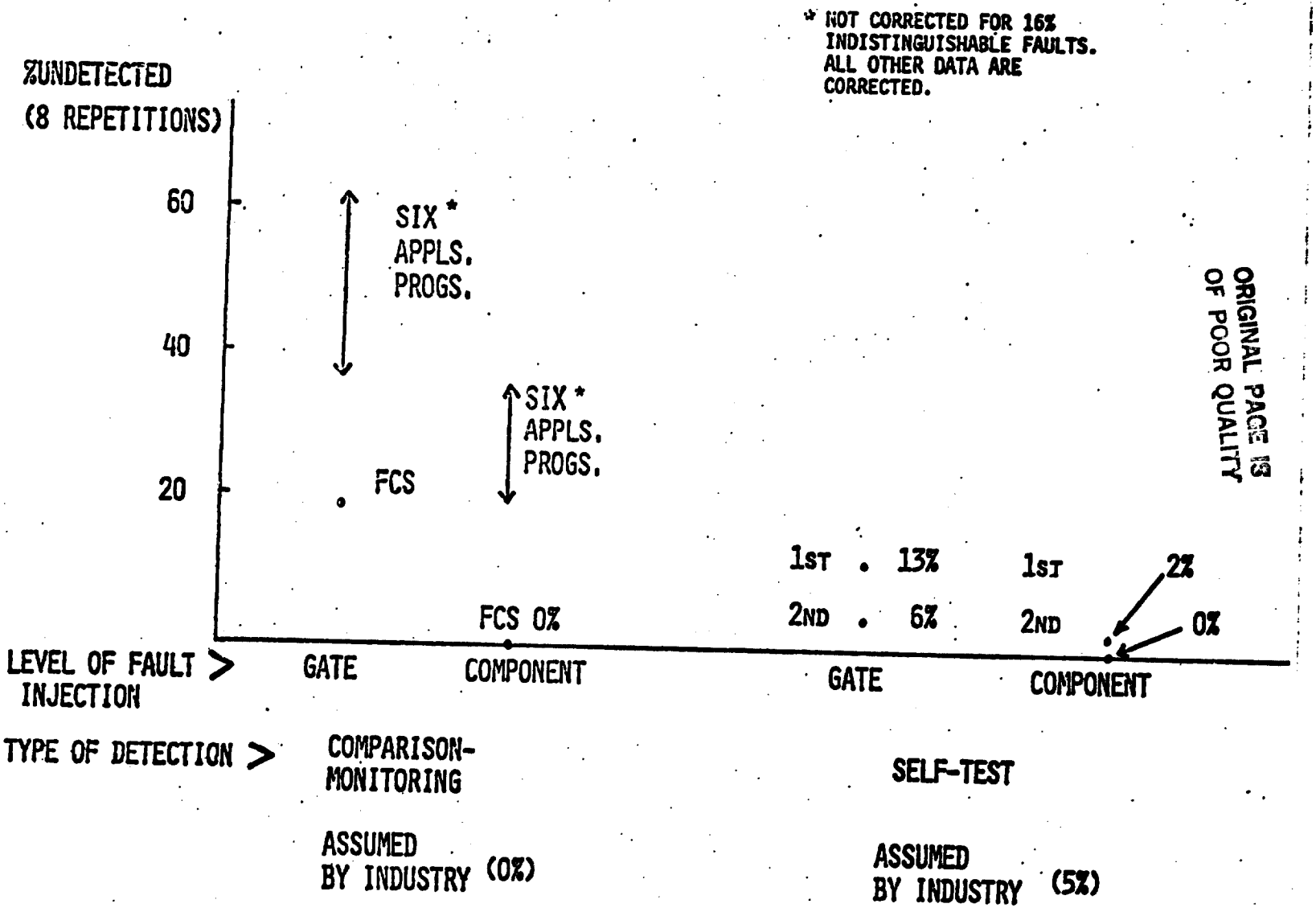


Figure 18. - Summary and results.